**What are the key differences between Kotlin and Java in terms of syntax and features?**

Hello everyone,

Java and Kotlin are very similar languages on the surface, which arguably makes their differences even more interesting to explore and discuss. Many of the syntax changes and features present in Kotlin aim to solve problems that developers had with programming in Java. For instance, Java does not entirely support functional programming – that is, functions in Java cannot be assigned to variables or used as return values (although the language does support lambda functions and some other features) (Digma Team, 2024). Kotlin completely supports functional programming, allowing for more concise and readable code in some circumstances.

Another interesting syntax feature of Kotlin for reducing errors is null safety. In Java, when a NullPointerException occurs due to attempting to access an object with a null value, the developer needs to handle the exception manually (Digma Team, 2024). Kotlin instead allows developers to specify that a particular variable is nullable using the “?” operator. This operator can also be applied when accessing a member of a variable, such as in the following statement:

val len: Int? = str?.length

In this case, if the value of “str” is null, instead of producing an exception, accessing the length member will return null. Notice that the “len” variable is also a nullable integer object.

Kotlin’s syntax is also simply more concise than Java’s in a lot of ways. A good example of this is defining a list. In Java, the programmer has to initialize the list explicitly and then either create a collection of values to add to the list all at once or add each element one-by-one. Kotlin instead allows for this syntax:

val list = listOf(“a”, “b”, “c”, “d”)

Having simple one-line code for definitions such as this makes the code much easier to understand.

On another note, there are some features of Java that Kotlin does not support. The ternary operator is a useful example here – Kotlin simply does not support it. Instead, developers are encouraged to use if/else statements, which can still be consolidated to one line (Kotlin, 2024). For programmers who prefer to use the ternary operator, this may be a downside. Kotlin also does not have pattern matching, instead implementing the “when” statement for more basic checks. Java’s pattern matching is much more exhaustive, albeit more difficult to read in some cases.

In general, Kotlin aims to be a more concise, less confusing language than Java and also to provide various desirable features that Java does not support. While switching from Java to Kotlin can mean giving up certain comfortable programming habits, it may enable many more useful techniques. As someone who programs primarily in Java and Python, I am interested to learn my thoughts on the language throughout this course.

References:

Digma Team. (2024, February 21). *Kotlin & Modern Java: The 17 Differences Experienced Developers Should Know*. Digma. <https://digma.ai/kotlin-modern-java-the-17-differences-experienced-developers-should-know/>

Kotlin. (2024, April 3). *Comparison to Java*. Kotlin Lang. <https://kotlinlang.org/docs/comparison-to-java.html>